**Advanced Computer Networking and Security**

**Project Report**

**Rajesh Podipati Anjeneyalu**

**700645933**

**Project Title: Simulation of Sliding Window Protocols**

**Preferred Background:** Programming language – C++

Communications Protocols

**Professor: Dr. Yiheng Liang**

Department of Mathematics and Computer Science

E-mail: [liang@ucmo.edu](mailto:liang@ucmo.edu)

**Project Synopsis:**

The objective of the project is to implement and research on two generic forms of pipelined approaches in transport service, go-Back-N (GBN) and selective repeat (SR), through software simulations.

**Project Contents:**

* Introduction to Sliding Window Protocol
* Go-Back-N Protocol
* Algorithm Associated with Go-Back-N
* Evaluation of performance for Go-Back-N
* Selective Repeat Protocol
* Algorithm Associated with Selective Repeat
* Evaluation of performance for Selective Repeat
* Comparison between Go-Back-N and Selective Repeat
* Conclusion of Go-Back-N and Selective Repeat
* Coding of the model

**Introduction:**

Sliding Window protocol is practically implemented in two ways.

Sliding Window Protocol

* Go Back N
* Selective Repeat

Reason for going Sliding Window protocol is pipelining. The Sliding Window protocol was developed to overcome the limitations of Stop-And-Wait flow control. In cases where the bit length of the link is greater than the frame length, efficiency can be greatly improved by allowing multiple frames to be in transit at the same time.

**Go-Back-N Protocol**

The sending process continues to send a number of frames specified by a window size without receiving an ACK packet from the receiver. If any frame was lost or damaged, or the ACK acknowledging them was lost or damaged, then that frame and all following frames in the window (even if they were received without error) will be re-sent. The *sender* maintains a buffer of a predetermined size. If there is room in the buffer it gets a packet, stores it in the correct empty slot (seq\_nr%WINDOWSIZE).

Makes an edge with the right seq\_nr and transmits it. The comparing coherent clock is then reset to 0. The Upper Bound of the window is then slid up by circularly increasing the next\_frame\_to\_send. As of right now or if no cushions are vacant, the physical layer is verified whether an affirmation arrives. On the off chance that a decent edge is gotten and the affirmation number is inside of the present window then decrements the quantity of supports utilized, reset the coherent clock (to a negative worth) to demonstrate an unused opening, and slide the Lower Bound of the window by circularly increasing the affirmation number anticipated. This technique keeps running in a circle until ack\_expected measures up to ack\_received (this clears the ack\_received and the past casings that haven't been recognized yet).

After this, or if a terrible casing or out of window edge arrives, the intelligent clocks are upgraded. In the event that a casing is timed out, it's retransmitted and the clock is reset to 0. In this style if an edge is timed out on the grounds that it was lost, no affirmation will arrive. So on the following emphasis of the circle, the following casing will time out and will be re-sent. In this manner the timed out edge and all the ensuing edges will be retransmitted, which is the meaning of Go Back N. The recipient occupied holds up until a casing arrives. In the event that an awful edge arrives, it backtracks to occupy with holding up. On the off chance that a decent edge arrives, it checks the succession number, in the event that it's not the grouping number expected it resends an ack for the last right arrangement number got. On the off chance that it's the succession number expected, it passes the bundle to the Network layer, redesigns the last right arrangement number got variable and circularly augments the following grouping number expected variable. An affirmation is then made and transmitted for it and after that it circles back to the physical layer to recover the following edge.

The cushion comprises of a variety of parcels. The extent of this cluster in the usage is 4, initially a variety of size 8 was utilized yet this came about as a part of issues. At that point a support size of 4 was utilized with grouping numbers 0-3, which still brought on issues. Dr. Yuan taught the class to utilize a Window size of 4 and grouping numbers bigger than the window. This execution utilizes succession numbers 0-7, as that is the thing that the reading material utilized and the littler scope of numbers is less demanding to work with. The Lower Bound of the window is spoken to by the ack\_expected and the Upp-++er Bound by the next\_frame\_to\_send.

In general, for a k-bit sequence number field:

Range of sequence numbers: 0  2k – 1

For Go-Back N, the maximum window size is: 23 – 1 = 7 frames

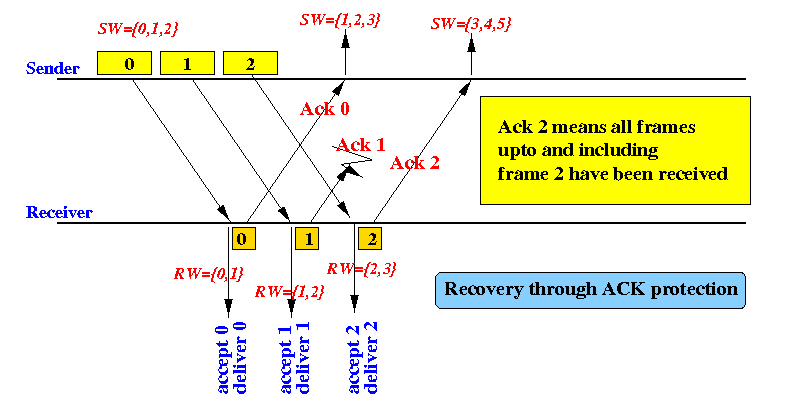
* Go-Back-N Example: Error Free Operation

|  |
| --- |
| http://www.mathcs.emory.edu/~cheung/Courses/558/Syllabus/01-ARQ/FIGS/window1.gif |

* + Notice how the sender and receiver windows change.
  + Specifically: notice how a sequence number is always removed the receiver window first, before it is removed from the sender window
* Go-Back-N Example: How the system recovers from a Sender Error

|  |
| --- |
| http://www.mathcs.emory.edu/~cheung/Courses/558/Syllabus/01-ARQ/FIGS/window2.gif |

* Go-Back-N Example: How the system recovers from a Receiver (ACK) Error - case 1: not the last ACK



* Go-Back-N Example: How the system recovers from a Receiver (ACK) Error - case 2: the last ACK

|  |
| --- |
| http://www.mathcs.emory.edu/~cheung/Courses/558/Syllabus/01-ARQ/FIGS/window3b.gif |

* Go-Back-N Example: How the system recovers from delayed ACK Error

|  |
| --- |
| http://www.mathcs.emory.edu/~cheung/Courses/558/Syllabus/01-ARQ/FIGS/window4.gif |

**Sender Algorithm for this protocol**

* Start.
* Establish connection.
* Accept the window size from the receiver (should be <=40)
* Accept the packets from the network layer
* Calculate the total frames/windows required.
* Send the details to the receiver (total packets, total frames.)
* Initialize the transmit buffer.
* Built the frame/window depending on the window size.
* Transmit the frame.
* Wait for the acknowledgement frame.
* Check for the acknowledgement of each packet and repeat the process from the packet for which the first negative acknowledgement is received. Else continue as usual.
* Increment the frame count and repeat steps 7 to 12 until all packets are transmitted.
* Close the connection.
* Stop.

**Receiver algorithm for this Protocol**

* Start.
* Establish a connection.
* Send the window size on sender request.
* Accept the details from the sender (total packets, total frames).
* Initialize the receive buffer with the expected packets.
* Accept the frame/window from the sender.
* Check for validity of the packets and construct the acknowledgement frame depending on the validity.(Here the acknowledgement is accepted from the users)
* Depending on the acknowledgement frame readjust the process.
* Increment the frame count and repeat steps 5-9 until all packets are received.
* Close the connection.
* Stop.

**Transmission delay:** Transmission delay (or store-and-forward delay, also known as packetization delay) is the amount of time required to push all the packet's bits into the wire. In other words, this is the delay caused by the data-rate of the link.

Transmission delay is a function of the packet's length and has nothing to do with the distance between the two nodes. This delay is proportional to the packet's length in bits,

It is given by the following formula:

D_T = N/R Seconds

Where

 - The transmission delay in seconds



N - The number of bits, and

R - The rate of transmission (say in bits per second)

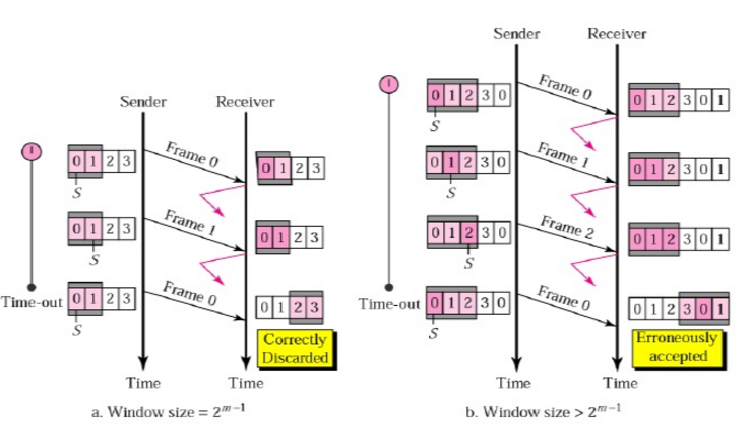
###### Performance

The ***Go Back N*** protocol is fairly tricky to implement. If the window size matches the range of sequence numbers the protocol can break and implementation is extremely difficult. By utilizing the scope of succession numbers more noteworthy than the window measure, the recipient and the sender can be kept in synchronization notwithstanding when edges and affirmations are lost at high rate. The Protocol effortlessly handles clog blunders, terrible casings and lost edges. Most troublesome part was attempting to locate the ideal timeout esteem that gave effective transmission to all blunder rates. Littler timeout esteem appeared to be better on higher blunder rates yet to upgrade at lower mistake rates a somewhat higher estimation of timeout was chosen. For e.g., at 40% mistake rates on all channels, timeout estimation of 10 gave the best result. By going marginally bring down at 8 the quantity of edges that must be transmitted expanded. By going marginally higher at 12, number of casings transmitted expanded all the more quickly. The high number of casings retransmitted implies waste as far as Bandwidth and a more extended time to finish the transmission. This is on account of for each lost edge every one of the edges transmitted after that casing are retransmitted.

**Selective Repeat Protocol**

The sending process continues to send a number of frames specified by a window size *even* after a frame loss. Unlike Go-Back-N ARQ, the receiving process will continue to accept and acknowledge frames sent after an initial error. If a frame from the sender does not reach the receiver, the sender continues to send subsequent frames until it has emptied its window. The receiver continues to fill its receiving window with the subsequent frames, replying each time with an ACK containing the sequence number of the earliest missing frame. Once the sender has sent all the frames in its window, it re-sends the frame number given by the ACKs, and then continues where it left off. The Selective Repeatprotocol improves on the Go Back N protocol by having buffers on both the sending and receiving sides. This allows the *sender* to have more than one outstanding frame at a time and *receiver* to accept out of order frames and store them in its window.

*Sender* for Selective Repeatis only slightly modified from that for Go Back N .The Maintenance of buffers and logical timers is exactly the same. The only difference is that if a negative acknowledgement is received, the sender retransmits the corresponding frame identified by the nak. Other than this timeouts, loop iterations and retransmissions are all the same as Go Back N. This differs from Go Back N in that it retransmits only the frame for which a nak is received and not all subsequent frames. As the receiver keeps a window of frames only the timed out frame needs to be retransmitted and not the whole series.



The recipient involved holds up until an edge arrives. In case a timeout happens or if a terrible edge arrives, or if an out of collection packaging arrives, and a nak has not been sent yet then a nak is sent for the typical course of action number. In case there is room in the recipient's support a package is secured in the right opening (progression number%WINDOWSIZE) and the space is hailed as used. Circle is ran starting at support opening for expected gathering number. If this opening is full the group is gone to the framework layer, a flag is set to send an attestation, pad space is reset to release, the upper bound of the window is extended, and the lower bound (frame\_expected) is circularly enlarged. It then circles back to check the pad opening for expected progression number. The circle continues till the ordinary space is void. Along these lines all supported bundles are gone to the framework layer all together. In case the flag that shows whether a confirmation must be sent or not (send\_ack) is set then an attestation is sent for the last right in progression diagram got. By then we retreat to the essential involved with holding up circle and start by and by yet again.

**Sender Algorithm for this protocol**

* Start.
* Establish connection
* Accept the window size from the receiver (should be <=40)
* Accept the packets from the network layer.
* Calculate the total frames/windows required.
* Send the details to the receiver (total packets, total frames.)
* Initialize the transmit buffer.
* Built the frame/window depending on the window size.
* Transmit the frame.
* Wait for the acknowledgement frame.
* Check for the acknowledgement of each packet and repeat the process for the packet for which the negative acknowledgement is received. Else continue as usual.
* Increment the frame count and repeat steps 7 to 12 until all packets are transmitted.
* Close the connection.
* Stop.

**Receiver Algorithm for this protocol**

* Start.
* Establish a connection.
* Send the window size on sender request.
* Accept the details from the sender (total packets, total frames).
* Initialize the receive buffer with the expected packets.
* Accept the frame/window from the sender.
* Check for validity of the packets and construct the acknowledgement frame  depending on the validity.(Here the acknowledgement is accepted from the users)
* Depending on the acknowledgement frame readjust the process.
* Increment the frame count and repeat steps 5-9 until all packets are received.
* Close the connection.
* Stop.

###### Performance

The Selective Repeat convention was hard to execute. The succession numbers should be more prominent than the window estimate so that no cover can happen in the window. This permits recipient and sender to be kept in synchronization notwithstanding when casings and affirmations are lost at a high rate. The buffering and affirmations permit this convention to effectively handle clog, awful edges and lost casings. It was found that a much higher timeout quality is required than in Go Back N keeping in mind the end goal to lessen the quantity of casings sent. A lower timeout esteem results in an excess of casings timing out and being retransmitted pointlessly, since the collector keeps up a cradle of edges and can send a nak for precisely the edge grouping number it needs. We have picked 25 as a timeout esteem, and kept the from\_physical\_layer parameter to 100,000 on the grounds that these two qualities worked the best for an extensive variety of blunder rates.

**Comparison:**

|  |  |
| --- | --- |
| **GO BACK AND ARQ** | **SELECTIVE REPEAT ARQ** |
| Go Back N ARQ is inefficient for noisy link. | Selective repeat ARQ is efficient for noisy links. |
| Go Back N ARQ is less complicated than Selective repeat ARQ. | Selective Repeat ARQ is complicated. |
| Go Back N ARQ Sender Window Size is 2^ (m)-1 and receiver window size is 1. | Sender and receiver Window Size is 2^ (m-1) |
| Go-Back-N ARQ is a particular case of the programmed rehash demand (ARQ) convention, in which the sending process keeps on sending various casings determined by a window measure even without getting an affirmation (ACK) bundle from the beneficiary. It is a unique instance of the general sliding window convention with the transmit window size of N and get window size of 1. | Selective Repeat ARQ/Selective Reject ARQ is a particular case of the Automatic Repeat-Request (ARQ) convention utilized for correspondences. It might be utilized as a convention for the conveyance and affirmation of message units, or it might be utilized as a convention for the conveyance of subdivided message sub-units. |

**CONCLUSION**

***Go Back N*** convention required more data as far as execution because of cradle support and keeping sender and recipient in synchronization. This convention is the most wasteful amongst the 4 because of the huge number of edges it retransmits. At whatever point an edge is lost as opposed to retransmitting the edge it retransmits every single ensuing casing likewise which squanders Bandwidth.

***Selective Repeat*** only took slight modification over ***Go Back N*** and produced far more efficient results in terms of the number of retransmissions, as it retransmits only one frames instead of the entire series.

**So overall in a communication channel with errors *Selective Repeat* would be a better choice**.